A Consistency Regulation Algorithm for Client-Server-based Multiplayer Computer Games

Séamus C. McLoone, Damian Wynne, Aaron B. McCoy and Tomás E. Ward

Department of Electronic Engineering,

National University of Ireland Maynooth,

Maynooth, Co. Kildare, Rep. of Ireland.

email: seamus.mcloone@eeng.nuim.ie; tomas.ward@eeng.nuim.ie

***Abstract*— Consistency is one of the most important aspects to be considered when designing a Distributed Interactive Application, and particularly in networked Multiplayer Computer Games (MCGs). Several techniques exist which aim to reduce network traffic in an attempt to maintain an acceptable level of consistency. However, these techniques are static in nature, as they do not adapt to the varying network conditions. This paper presents an algorithm which aims to maintain an acceptable level of consistency by adapting to changes in the network. The algorithm is a rate-based approach which monitors the remote inconsistency of all players participating in a client-server-based MCG. It operates on the premise that as the network conditions across the links between the server and the clients vary, so too will the remote inconsistency. In response, the rate at which the updates are transmitted from the server to all the remote clients is adapted, ensuring that an acceptable level of consistency is maintained. Simulation results for the proposed algorithm are also presented.**

***Keywords* – Consistency, Client-Server Network Architecture, Multiplayer Computer Games, Distributed Interactive Applications.**

I INTRODUCTION

A Distributed Interactive Application (DIA) is a computer program which enables geographically dispersed users to act in real-time in a simulated environment to perform interactive tasks [1, 2]. Over the past three decades, three distinctive classes of DIAs have come to the fore, namely military simulations, networked virtual environments and online multiplayer computer games (MCGs) [2].

Due to the rapid advancements in online gaming technology, it is the MCGs that have seen the greatest increase in demand and popularity in recent years, with games such as the *Massively Multiplayer Online Game* (MMOG) hosting over a thousand players at any instance in time [3]. MCGs present their own unique challenges as they require high consistency and high responsitivity. Players of MCGs expect their actions to be interpreted immediately and they assume that the representation of the virtual world which they are viewing is faithful and true. The distributed nature of MCGs and DIAs in general, presents several obstacles to meeting these expectations, as game designers have to overcome challenges presented by real world communication networks, most commonly the Internet. These include limited bandwidth and network latency.

In reality, perfect consistency is unachievable. There is a balance required between the number of updates transmitted among hosts and the actual consistency obtained. On one hand, if too few updates are sent then the consistency is greatly reduced. On the other hand, if too many updates are sent, the bandwidth is flooded, updates are delayed and consistency is again reduced. This concept is known as the consistency-throughput trade-off [4].

To date, current MCGs employ static methods for sending updates, such as dead reckoning models [5] with fixed error thresholds or simple fixed rate-based transmission. However, these methods do not allow for changing network conditions and therefore cannot guarantee or maintain an acceptable level of consistency. Some work has been carried on an adaptive entity state update technique, but this has primarily focused on peer-to-peer DIAs [6].

This paper proposes a novel global consistency-aware entity state update algorithm for a client-server MCG. This algorithm is a rate-based technique that monitors a measure of inconsistency recorded by each remote client and adjusts the rate at which the remote clients receive updates from the server accordingly. This new technique is simulated using the NS2 network simulator [7] and its performance is evaluated in comparison to a standard fixed rate method.

The rest of this paper is structured as follows. The next section describes two of the key aspects associated with MCGs, namely consistency and responsitivity. It also describes how the various communication layers within networked games

affect both of these aspects. Section III outlines the proposed rate-based method for entity state updates in client-server MCGs. Section IV describes the simulation used to evaluate the performance of the proposed algorithm. The NS2 simulator is also described here. The results of the simulation are presented and discussed in section V, while the paper ends with conclusions in section VI.

II CONSISTENCY & RESPONSITIVITY

Consistency in a MCG refers to the ability of the MCG to ensure that each user's view of the world is identical, or as close to identical as can be achieved for given conditions. A basic aspiration for a distributed simulation is to present entities as being *in the right place at the right time* and a simple metric to capture this notion is a spatially-derived time-stamped measure of consistency that can be calculated using distance measures between the entity positional state at its local peer and its representations at remote peers.

The responsitivity of a MCG refers to how quick an action taken by a user, in the form of a physical key press or a joystick movement, is processed and played out in the virtual world. In the case of a client-server architecture (see Figure 1), such an action has to be sent to the server for processing and verification. For example, consider the case where a player strikes the mouse button in order to lift an object in an environment before another opponent. This event is then sent to the server, where it is processed and verified. Once the server has verified that this is a valid operation, in this case that the opponent has not lifted the object first, a response is sent back to the player indicating that they are allowed to take the object. The time it takes for this to occur is known as the response time.

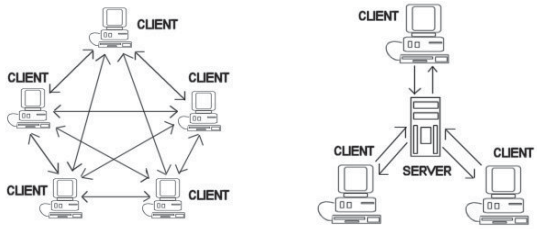
a) The physical and logical platforms

Multiplayer computer games can generally be broken down into three separate communication layers, namely the physical platform (this refers to the underlying network infrastructure), the logical platform (this builds upon the physical platform and provides architectures for communication, data, and control) and the networked application (this provides for the interpretation of the data). Of these, it is the physical and logical platforms that have the greatest affects on consistency and responsitivity.

The physical platform for any distributed application imposes certain restrictions on the speed and reliability of the communication of data. These issues are caused by resource limitations including the network bandwidth, the network latency and the processing power of each node for handling the network traffic. These limitations are tightly coupled with the consistency and responsiveness of the application. For MCGs, as previously mentioned, both high responsiveness and high consistency are important. In reality, it is difficult to have an MCG which is both highly responsive and highly consistent. The physical platform dictates that only a trade-off between these aspects can be achieved.

There are many existing techniques which aim to compensate for the limitations imposed by the physical platform of the network. These techniques tend to rely on the principle of message reduction, in an effort to prevent overloading of the network. However, on the other hand, an abundance of messages can lead to packet loss and increases in network latency, which adversely affects both responsiveness and consistency.

The logical platform is concerned with the flow of traffic in the network. As such, it defines the communication, data, and control architectures. There are a number of models which can be chosen for the communication architecture, the two most commonly used being client-server and peer-to-peer.



(a) (b)

Figure 1: Communication models - (a) peer-to-peer and (b) client-server.

b) Peer-to-peer & client-server

In the peer-to-peer model, as shown in Figure 1(a), each node is equal, and may transmit messages to any other node in the network. Early networked games relied on this peer-to-peer model as it is fairly straightforward to realize. Also, since each peer is equal and maintains authority over its own state, high responsitivity can be assured as the players actions may be interpreted immediately by the node. This comes at the cost of consistency, as conflicts can arise over the true game state due to the lack of a single controlling entity. Problems can arise when attempting to determine ownership of a particular item, or determining if a player has successfully seized an item before an opponent, for example.

The client-server model, as illustrated in Figure 1(b), works by introducing a server node into the network. All nodes in the network, called clients, need only be concerned with connecting to this server. They have no interest in any of the other clients in the network, as the server is responsible for all communication between each of them. In this way the server can monitor all traffic between clients and maintains the authoritative state of the game world. Thus, high consistency can be maintained, but at the cost of the responsitivity of the application, as all user input must first be approved by the server before being acted on, introducing delays between the time a player initiates an action and the time it takes place.

Several different entity state consistency management methods exists for reducing network traffic while maintaining some level of consistency [5, 8]. However, these methods are static in nature and do not adapt to changing network conditions, such as increases in latency or reduction in available bandwidth. There has been recent investigation into using adaptive techniques based on the inconsistency at remote users [6]. However, this work has focused on peer-to-peer DIAs to date. Nevertheless, it provides inspiration for the development of an adaptive rate-based entity state update method for use in client-server MCGs. This proposed method is now presented.

III CLIENT-SERVER CONSISTENCY REGULATION (CSCR) ALGORITHM

The proposed algorithm strives to maintain an acceptable, preset level of inconsistency by adjusting the update transmission rate from the server to the client in accordance with the current level of remote spatial inconsistency at that client. This rate-based approach is dependent on two important aspects, namely the required desired level of inconsistency and a measure of the remote inconsistency.

a) Desired level of inconsistency (DLI)

As noted in section II, absolute consistency is not achievable. As such, we need to choose an acceptable level of inconsistency and allow the proposed algorithm to maintain this level. Choosing this value is not trivial and will need to be carefully considered at design time, using knowledge of the network environment, the type of MCG interaction supported and the level of acceptable error. Ideally, it should be a value that can be maintained with a low transmission rate in order to avoid excess traffic generation, while also ensuring the inconsistency produced is not perceptually noticeable. In addition, the rate should be able to increase in the presence of increased latency without causing the traffic to exceed the bandwidth limitation. By way of illustration, consider the graph in Figure 2. This graph shows how inconsistency (pixels) varies with the transmission rate (packets per second) for an arbitrary simulation example. The relevant detail to note here is that the bandwidth limit occurs at a transmission rate of 35pps. Three different DLI values are also shown.

In this case, a DLI of 0.5 pixels is too low as the inconsistency is never reduced to that level, even at the maximum possible transmission rate before exceeding the bandwidth limitation. This would cause the algorithm to behave as though it were attempting to achieve absolute consistency. In contrast, a DLI of 20 pixels can be maintained at a very low transmission rate. However, it is obvious that by increasing the rate by a very small amount the inconsistency can be maintained at a much lower level, which is clearly preferable. In Figure 2 (b) it is observed that a DLI of 5 pixels can also be maintained at low transmission rates but, more importantly, it takes a relatively larger increase in transmission rate to reduce the level of inconsistency experienced. Hence, this is a good choice of DLI for this particular simulation. A lower value may also be chosen along this curve, but as the DLI gets lower, the transmission rate required gets closer to the bandwidth limitation, reducing the scalability of the application, as well as the additional latency that can be accounted for.

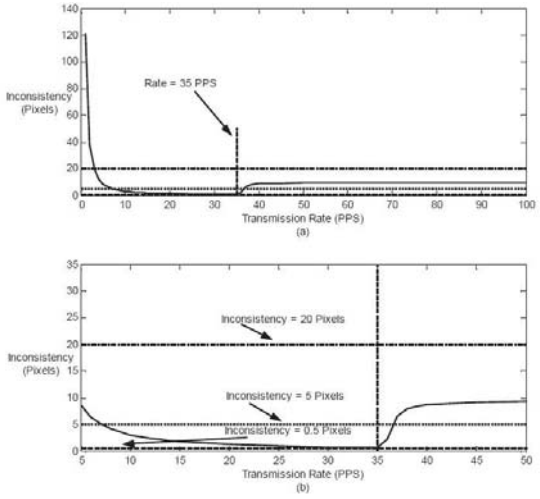


Figure 2: (a) Sample inconsistency profile with example DLI levels; (b) a magnified version of same.

It is worth noting, from Figure 2 (b), that a DLI of 5 pixels provides two points along the inconsistency curve. The first is around which the inconsistency is to be maintained (the DLI) and the one at the lowest transmission rate. The other point occurs where the inconsistency is increasing. This is the point where the transmission rate has caused the traffic on the network link to exceed the available bandwidth and is the point with the largest transmission rate. Clearly, the latter point needs to be avoided. These two points can be easily distinguished through monitoring packet loss, which increases when the bandwidth is exceeded. By detecting the instances when the traffic exceeds the available bandwidth it then becomes possible to reduce congestion on the network. This is achieved by increasing the DLI and reducing the transmission rate, maintaining a higher level of inconsistency, but guaranteeing that higher level.

b) Measuring remote inconsistency

The remote inconsistency measured and used by the proposed algorithm is the error due to latency in addition to the time between updates. This error is best illustrated with the aid of an example, as shown in Figure 3. This figure shows the path an object may take. Along this path, two updates are generated. The first is generated at time t = 0, with position $\mathbf{p_1}$ and velocity vector $\mathbf{v_1}$, as shown. At a time t_u later the

second update is generated with position $\mathbf{p}_2$ and velocity vector $\mathbf{v}_2$. The remote client does not receive this new update until a further time τ later, indicating the delay experienced. The remote inconsistency e_r is calculated as the spatial error at this point in time.

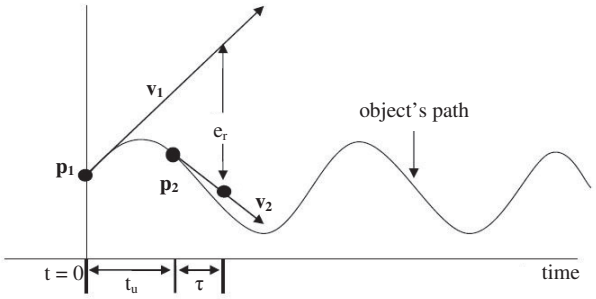


Figure 3: Measuring remote inconsistency e_r.

c) Ignoring sudden spikes in inconsistency

Sudden short-lived spikes can sometimes occur in remote inconsistency for reasons such as packet loss or extremely acute changes in entity motion. These spikes need to be ignored otherwise they will cause a sudden increase in transmission rate followed by an almost immediate reduction, resulting in unnecessary usage of the bandwidth. In order to reduce the sensitivity of the proposed algorithm to such spikes, rate changes are only generated when the DLI is exceeded for a minimum period of time, defined here by the parameter T_{DLI}.

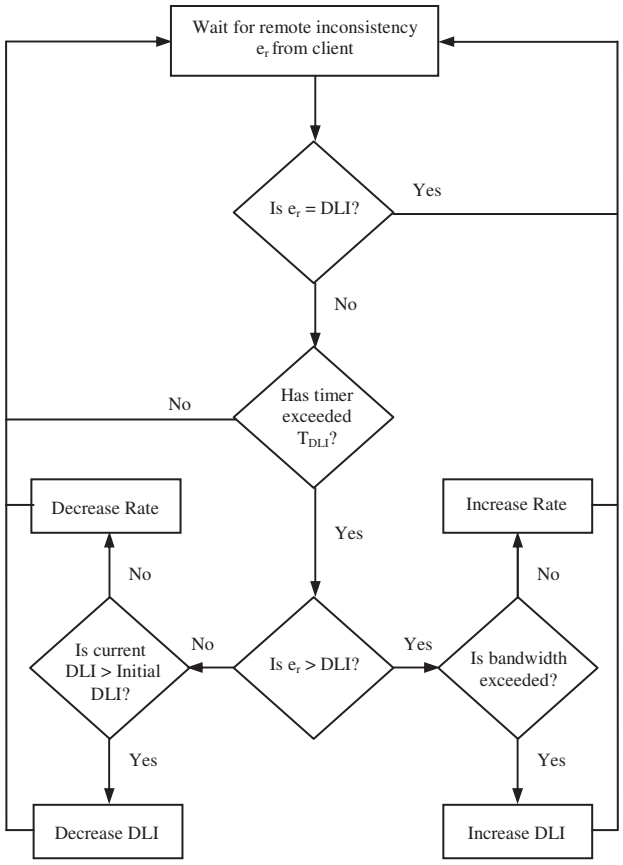


Figure 4: Flowchart of the CSCR algorithm.

d) The CSCR algorithm

Figure 4 outlines the actual CSCR algorithm as implemented by the server in a client-server scenario. In brief, the server receives a measure of the inconsistency from the remote client, ensures that any changes in this value exists for a short, but sustained period of time (by using a timer) and, subsequently, changes the transmission rate of packets to the same client accordingly. Figure 4 also shows that the DLI increases if the bandwidth is exceeded and once network congestion eases, the DLI value can return to its original value once again. This is to ensure that the lowest possible DLI is maintained at all times.

It is worth noting that if the inconsistency is less than the DLI then the transmission rate is decreased. Leaving the rate unchanged would result in a lower level of inconsistency. However, this occurs at the expense of additional traffic being generated. By maintaining the level inconsistency at the preset DLI, there is more bandwidth available, and thus the potential for allowing more users of the application, or possibly even allowing other data types to be transmitted along with the updates that would not have been otherwise possible.

Finally, as acceptable levels of consistency may be impossible to achieve due to excessive traffic, or even network failure, limits must be set on both the server transmission rate and the DLI. Under such severe circumstances, a solution is to remove any clients which cannot maintain the acceptable level from the MCG. Each of these limits are important and are generally application specific.

IV SIMULATING THE CSCR ALGORITHM

This section outlines the simulation setup used to evaluate the performance of the proposed CSCR algorithm. Firstly, the NS2 simulator used to simulate the algorithm is briefly presented.

a) The NS2 simulator

NS2 is an event driven network simulator developed at UC Berkeley [7]. It is targeted at networking research that simulates a variety of Internet Protocol networks. It implements network protocols such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), along with numerous traffic source behaviours, router queue management mechanisms, routing algorithms, and more. It also implements multicasting and some of the MAC layer protocols for LAN simulations. NS2 is developed using both the C++ and OTcl programming languages. The simulator itself is written in C++ with the command and configuration interface being implemented using OTcl.

b) Simulation setup

In order to test the CSCR algorithm some means of simulating the motion of an object was required. Three types of motion were simulated,

namely *smooth*, *bounce* and *jolt*, as these represent the curve classifications by which the motion of an object can be locally described [9].

The smooth motion is simply represented by a circular path. The bounce motion is used to simulate a sudden change in direction, i.e. an object which is travelling in one direction and suddenly turns and begins travelling in the opposite direction. This is implemented by causing the object to reverse its direction once it reaches a certain point, for example where a ball comes into contact with the ground. The jolt motion is used to simulate an object moving in a random path and is obtained by introducing a second circle to the smooth motion. A visual representation of the jolt motion is given in Figure 5.

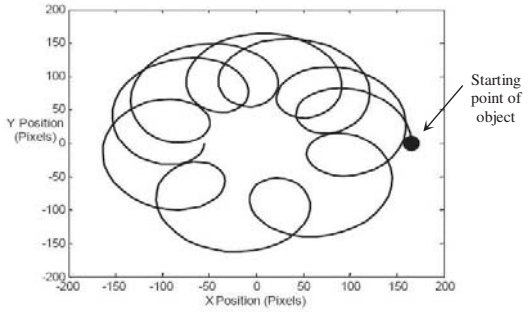


Figure 5: Simulated jolt motion.

In addition to simulating these motions, the parameters of the CSCR algorithm had to be chosen. Here, the initial DLI value is 5 pixels and the initial transmission rate is 5pps. If required, the DLI will be increased by an increment of 5 pixels and reduced by a factor of 10%. In general, these parameters need to be carefully chosen at design time and are application dependent.

In addition, it was decided to adopt four different transmission rate changes. This allowed the algorithm to remain highly responsive and, also, introduced the ability to fine-tune the transmission rate as the inconsistency approaches the DLI value. The values chosen here are as follows: if the remote inconsistency is < 70% or > 105% of the DLI, then the rate change is set to 1pps; if it is between 70% and 85% of the DLI then the rate change is 0.1pps; if is between 85% and 95% of the DLI then the rate change is 0.02pps; if it is between 95% and 105% of the DLI then there is no rate change.

c) Test Cases

Several tests were successfully simulated in order to illustrate the effectiveness of the CSCR algorithm. Due to lack of space, only two of these test cases are presented here. The reader is referred to [10] for a detailed set of analysis and results. Test Case 1 simply involves simulating a single client-server scenario where a server-side object follows each of the three motions outlined earlier. The client side views a simple linear extrapolation of the object's motion, as illustrated in Figure 3. The network latency for this test case was varied from 50ms to 250ms, increasing 50ms every 10s. The simulation ran for 60s. For the final 10s, latency is returns to 50ms. Test Case 2 involves multiple clients (4 in this case) and only the smooth motion is now considered. Here, each client is subject to a different value of latency, ranging between 100ms and 250ms. This simulation ran for 30s. Both test cases were carried out with CSCR and without CSCR. In the latter case, this simply means that a fixed transmission rate of 5pps is used throughout the simulation. The results of these test cases are now presented.

V RESULTS AND DISCUSSION

Figure 6 shows the inconsistency obtained for each of the motion types smooth, bounce and jolt for test case 1. The advantages of the CSCR algorithm are clearly evident, particularly in the case of the smooth motion, where the inconsistency is maintained around the DLI despite the changing latency. It is worth observing that slight 'bumps' occur in the inconsistency each time the latency changes. This is due to the increased latency experienced, and is corrected by adjusting the transmission rate. The transmission rate from the server to the client increases in order to compensate for the larger inconsistency and decreases whenever the latency is reduced. By adjusting the transmission rate, based on the inconsistency measured, the inconsistency can be regulated at the desired level. When the CSCR algorithm is not employed, the smooth motion simply experiences an inconsistency which mimics the changes in latency, due to the fixed transmission rate.

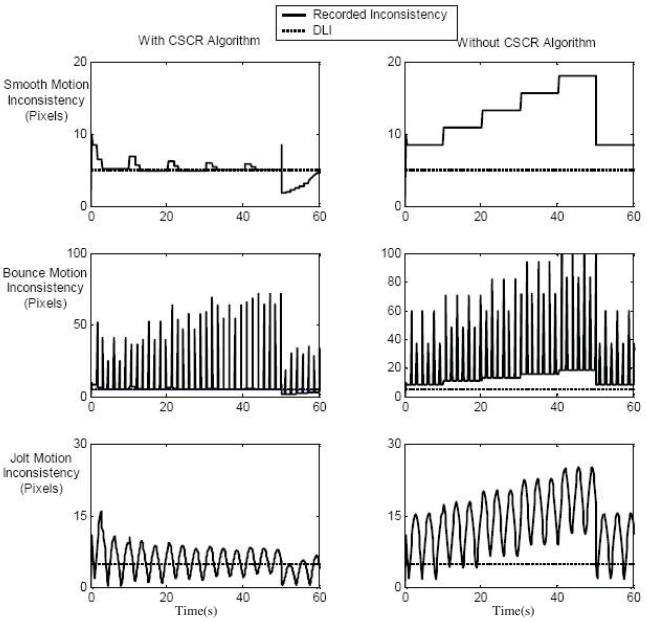


Figure 6: Inconsistency for test case 1 (smooth, bounce and jolt motions) with and without CSCR.

It is worth noting that when the latency is reduced at 50 seconds, a momentary spike in the inconsistency occurs. This is caused by packets being received out of order. In NS2, each packet transmitted has a delay associated with it. Once the latency changes, packets that are already transmitted do not change this value, experiencing longer delays than any newly transmitted packets. Hence, these packets are received by the recipient later than packets transmitted after them with a lower latency.

Similar observations can be made for the bounce and jolt motions in relation to an improved performance using the CSCR algorithm. Here, however, we can observe spikes in inconsistency for the bounce motion and constantly changing inconsistency for the jolt motion. These occur because of the simple client-side linear extrapolation employed. For example, this extrapolation predicts the ball travelling past the bounce point rather than turning at it and, hence, each time the ball bounces a spike in inconsistency occurs.

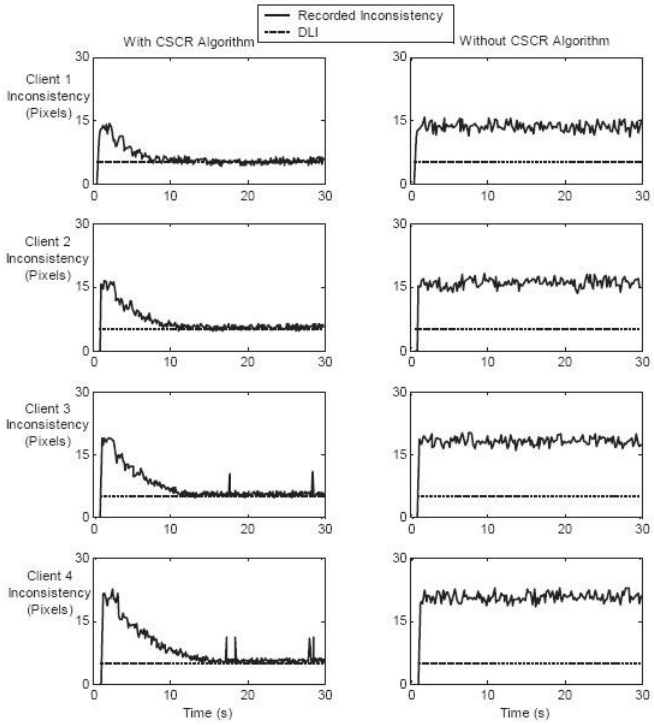


Figure 7: Inconsistency for test case 2 (4 clients with different latency values) with and without CSCR.

Figure 7 shows the inconsistency obtained for each of the four clients for test case 2. These results show that, despite differing levels of latency between each of the connected clients, the CSCR algorithm can adjust the transmission rate of each client to ensure that a uniform level of inconsistency is maintained. This is significantly better then the scenario in which a fixed transmission rate is employed. In this case, each client experiences a different level of inconsistency. Furthermore, due to the fixed rate, poor inconsistency values are not dealt with. The only hope for improving consistency is that network traffic from other applications reduces to increase available bandwidth.

It is important to note, that using the CSCR to maintain a uniform level of inconsistency for all clients may not always be a good idea, or even possible. As previously stated in section III, part (d), specific clients may need to be completely removed from the application for the 'greater good' of the remaining participants.

VI CONCLUDING DISCUSSION

This paper has proposed a novel consistency maintenance algorithm for use in client-server based MCGs and, indeed, DIAs in general. It operates by monitoring inconsistency levels at each client and adjusts the transmission rate to maintain this inconsistency at a preset level. The simulation results presented within validate this approach and show that the algorithm performs better than the current static method of employing a constant transmission rate throughout.

Future work will evaluate the performance of the proposed algorithm in more realistic distributed interactive virtual environments.

VII REFERENCES

[1] M. R. Macedonia and M. J. Zyda, "A Taxonomy for Networked Virtual Environments", *IEEE Multimedia Systems' 98*, 4(1), 1997, pp. 48-56.

[2] D. Delaney, T. Ward and S. McLoone, 2006, "On Consistency and Network Latency in Distributed Interactive Applications: A Survey - Part 1", *Presence: Teleoperators and Virtual Environments*, vol. 15, 2006, pp. 218-324.

[3] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi and E. Yuk, "Traffic Characteristics of a Massively Muli-player Online Role-Playing Game", *4th ACM SIGCOMM workshop on Network and System Support for Games*, New York, USA, 2005.

[4] S. Singhal and M. Zyda, "*Networked Virtual Environments: Design and Implementation*", New York ACM Press/Addison-Wesley Publishing Co., 1999.

[5] "IEEE Standard for Distributed Interactive Simulation - Application Protocols", 1998, *IEEE Std 1278.1a-1998*.

[6] D. Marshall, S. McLoone and T. Ward, "Optimising Consistency by Maximizing Bandwidth Usage in Distributed Interactive Applications", *ACM Transactions on Multimedia Computing, Communications, and Applications,* vol. 6, no. 4, article 30, 2010.

[7] K. Fall and K. Varadhan, K, "The ns Manual", available at http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

[8] A. McCoy, T. Ward, S. McLoone and D. Delaney, "Multistep-ahead neural network predictors for network traffic reduction in distributed interactive applications," *ACM Transactions on Modeling and Computer Simulations*, vol. 17, no.4, article 16, 2007.

[9] S. K. Singhal, "Effective Remote Modelling in Large-Scale Distributed Simulation and Visualization Environments," Dept. of Computer Science, Stanford University, CA, USA, 1996.

[10] D. Wynne, "Design of a Consistency Regulation Algorithm for Client/Server Based Multiplayer Computer Games", M.Eng.Sc. Thesis, Dept. of Electronic Engineering, NUI Maynooth, Ireland, 2009.